Activity 1. [Make a table reflecting the execution times of the PythonA1.py module for the exposed values of n (10000, 20000, 40000, 80000, 160000, 320000, 640000). If for any n it takes more than 60 seconds you can indicate OoT (“Out of Time”), both in this section and in subsequent sections]

|  |  |
| --- | --- |
| n | PythonA1.py time |
| 10000 | 3618 |
| 20000 | 13743 |
| 40000 | 54386 |
| 80000 | OoT |
| 160000 | OoT |
| 320000 | OoT |
| 640000 | OoT |

Activity 2. [Make a table that reflects, at least for two computers to which you have access, the execution times of the PythonA1.py module for the exposed values of n (10000,20000, …, 640000). Clearly indicate which CPU and RAM memory you are using in each test.]

|  |  |  |
| --- | --- | --- |
| n | Ryzen 5 3400G – 16GB RAM | I7 10870H – 16GB RAM |
| 10000 | 3618 | 4343 |
| 20000 | 13743 | 17474 |
| 40000 | 54386 | OoT |
| 80000 | OoT | OoT |
| 160000 | OoT | OoT |
| 320000 | OoT | OoT |
| 640000 | OoT | OoT |

Activity 3. [Program a class named JavaA1.java, which uses the same A1 algorithm to find out if a number is a primer number, as in PythonA1.py. Then, a table must be made reflecting the execution times of JavaA1.java for the same values of n (10000, 20000, ..., 640000). Finally, compare these times with those obtained in Python (in a previous section) for that same algorithm A1.]

|  |  |  |
| --- | --- | --- |
| n | PythonA1.py | JavaA1.java |
| 10000 | 3618 | 196 |
| 20000 | 13743 | 773 |
| 40000 | 54386 | 3051 |
| 80000 | OoT | 12502 |
| 160000 | OoT | 47563 |
| 320000 | OoT | OoT |
| 640000 | OoT | OoT |

The values for python are greater than for java, because different languages have different performance and the more low-level a language is, the faster.

Activity 4. [Make a table reflecting the execution times of the modules PythonA1.py, PythonA2.py and PythonA3.py, for the same values of n (10000, 20000, …, 640000). Codify those same algorithms A2 and A3 in Java, in two classes named respectively JavaA2.java and JavaA3.java. Make a table that reflects the execution times of the classes JavaA1.java, JavaA2.java and JavaA3.java, for the values of n (10000, 20000, ..., 640000) WITHOUT OPTIMIZATION of the Java program. Make a table that reflects the execution times of the classes JavaA1.java, JavaA2.java and JavaA3.java, for the values of n (10000, 20000, ..., 640000) WITH OPTIMIZATION of the Java program. Finally, draw final conclusions by comparing the times previously obtained: with Python, with Java WITHOUT OPTIMIZATION and with Java WITH OPTIMIZATION. Optionally, an A4 algorithm can be implemented in Java, which should improve the previous algorithms]

|  |  |  |  |
| --- | --- | --- | --- |
| n | PythonA1.py | PythonA2.py | PythonA3.py |
| 10000 | 3618 | 338 | 205 |
| 20000 | 13743 | 1309 | 650 |
| 40000 | 54386 | 5045 | 2453 |
| 80000 | OoT | 18307 | 9015 |
| 160000 | OoT | OoT | 33460 |
| 320000 | OoT | OoT | OoT |
| 640000 | OoT | OoT | OoT |

WITHOUT OPTIMIZATION

|  |  |  |  |
| --- | --- | --- | --- |
| n | JavaA1.java | JavaA2.java | JavaA3.java |
| 10000 | 546 | 60 | 40 |
| 20000 | 2033 | 207 | 151 |
| 40000 | 7058 | 788 | 506 |
| 80000 | 27337 | 2840 | 1804 |
| 160000 | OoT | 10572 | 6492 |
| 320000 | OoT | 38126 | 24960 |
| 640000 | OoT | OoT | OoT |

WITH OPTIMIZATION

|  |  |  |  |
| --- | --- | --- | --- |
| n | JavaA1.java | JavaA2.java | JavaA3.java |
| 10000 | 197 | 27 | 21 |
| 20000 | 742 | 104 | 61 |
| 40000 | 3032 | 300 | 181 |
| 80000 | 11793 | 1188 | 717 |
| 160000 | 50802 | 4357 | 2648 |
| 320000 | OoT | 18578 | 8897 |
| 640000 | OoT | OoT | 35428 |

Python is the slowest option. Regarding java, it can be seen that both the algorithm implemented and the compiler optimizations relevant in execution times, since optimized java is quite faster than non optimized java.